
AFRL-RI-RS-TR-2018-037

# INFERENCE BUILDING BLOCKS

INDIANA UNIVERSITY

*FEBRUARY 2018*

FINAL TECHNICAL REPORT

***APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED***

STINFO COPY

**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

# NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2018-037 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

**/ S /**
WILMAR W. SIFRE
Work Unit Manager

**/ S /**
JOHN D. MATYJAS
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

# REPORT DOCUMENTATION PAGE

*Form Approved*
**OMB No. 0704-0188**



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| FEB 2018 | FINAL TECHNICAL REPORT | OCT 2013 – NOV 2017 |

**4. TITLE AND SUBTITLE**

INFERENCE BUILDING BLOCKS

**5a. CONTRACT NUMBER**
FA8750-14-2-0007

**5b. GRANT NUMBER**
N/A

**5c. PROGRAM ELEMENT NUMBER**
61101E

**6. AUTHOR(S)**
Carette, Jacques; Kiselyov, Oleg; Mohammed Ismail, Wazim; Narayanan, Praveen; Ramsey, Norman; Romano, Wren; Scherrer, Chad; Shan, Chung-chieh; Smith, Geneva; Sullivan, Zachary; Toporovskyy, Yuriy; Walia, Rajan; Zinkov, Robert

**5d. PROJECT NUMBER**
PPML

**5e. TASK NUMBER**
2I

**5f. WORK UNIT NUMBER**
NU

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Indiana University
700 N Woodlawn Ave
Bloomington, IN 47408-3901

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

**10. SPONSOR/MONITOR'S ACRONYM(S)**
AFRL/RI

**11. SPONSOR/MONITOR'S REPORT NUMBER**
AFRL-RI-RS-TR-2018-037

**12. DISTRIBUTION AVAILABILITY STATEMENT**
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

We address the problem that probabilistic inference algorithms are diÿcult and tedious to implement, by expressing them in terms of a small number of building blocks, which are automatic transformations on probabilistic programs. On one hand, our curation of these building blocks reflects the way human practitioners discuss probabilistic inference with each other, so our probabilistic programming language supports modular composition of inference procedures and serves as a medium for collaboration. On the other hand, our implementation of these building blocks combines high-level mathematical reasoning with low-level computational optimization, so the speed and accuracy of the generated solvers are competitive with state-of-the-art systems.

**15. SUBJECT TERMS**
Generative probabilistic programs, modular Bayesian inference, composable program transformations, conditioning, disintegration, Markov chain Monte Carlo, exact inference on continuous distributions

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | WILMAR W. SIFRE |
| U | U | U | UU | 16 | 19b. TELEPHONE NUMBER (*Include area code*) |

## TABLE OF CONTENTS



## LIST OF FIGURES



## LIST OF TABLES

## 1.0 SUMMARY

The language we designed to underpin our approach is called Hakaru. Hakaru is relatively simple because it is concerned only with expressing random choices and not with expressing inference techniques. The expressions of Hakaru are *monadic*—basically, each expression is a sequence of random variable bindings followed by a final outcome term—and composed of primitive distributions. The primitive distributions in Hakaru include:

- the usual primitive distributions, such as Gaussian, Beta, Bernoulli, Categorical, as well as Dirac distributions;
- unnormalized distributions, such as the Lebesgue measure and the scaling of any measure by a weight; and
- random arrays generated by independent choices, popularized by so-called plate notation, which can be used to express Dirichlet distributions.

Hakaru enjoys an *operational semantics*, in the sense that every program can be executed as (or compiled to) a sampling procedure that on each run produces a sample outcome along with a non-negative weight. In particular, if a program does not use any unnormalized primitive distributions, then the weight is always 1 so each run of the sampling procedure simply produces a sample outcome.

Hakaru also enjoys a *denotational semantics*, in the sense that every program denotes a measure (more precisely, a function from inputs to s-finite measures). In particular, if a program does not use any unnormalized primitive distributions, then it denotes a probability measure.

What is innovative about our approach is that we express inference techniques not as implementations of the language but as transformations that take programs as input and produce programs as output. This approach makes it easier to compose and alternate inference techniques, in terms of a few simple building blocks:

- *Expectation* turns a given probabilistic program and a given function into an expression for the expectation of the given function with respect to the measure denoted by the given program.
- *Simplification* turns a given probabilistic program, which denotes a measure, into a more efficient program that denotes the same measure.
- *Disintegration* turns a given probabilistic program, which denotes a joint measure, into a program that denotes a *disintegration* of that measure. Roughly speaking, a disintegration is a possibly unnormalized *conditional* distribution of some random variables given others.

Using these building blocks, we have expressed a variety of inference techniques on discrete and continuous distributions: exact inference, importance sampling, Metropolis-Hastings (MH) sampling, Gibbs sampling, and slice sampling.

Because Hakaru is such a simple language, it is a well-suited medium for high-level mathematical reasoning as well as low-level computational optimization. Thus, our approach shines in application domains that call for both. One such domain is classification, whether unsupervised (such as clustering) or supervised (such as Naive Bayes). We observed the following advantages:

- A popular approach to classification is *collapsed Gibbs sampling* [1], and our approach makes it easy to express collapsed Gibbs sampling by composing Gibbs sampling and exact inference.

- Because Hakaru programs are mathematical expressions without any side effect such as mutation, they are easy to compile to efficient (in particular, parallel) machine code.

Both of these advantages are made much more powerful by the support for arrays in our language and transformations. Consequently, the classifiers generated using Hakaru are faster and more accurate than state-of-the-art probabilistic programming systems such as Just Another Gibbs Sampler (JAGS).

Our initial success leads us to recommend extending the Hakaru language to support more data types, extending the transformations to handle more distributions, and adding new transformations such as optimization and reparameterization.

## 2.0 INTRODUCTION

Our research aims to make probabilistic inference algorithms easier to compose. In Figure 1 and the rest of this section, we explain our research in relation to DARPA's Probabilistic Programming for Advancing Machine Learning (PPAML) program and other approaches to probabilistic programming.

### 2.1 Background

Probabilistic programming is a new approach to managing uncertain information that decouples and *separately* automates the tasks of developing a probabilistic model and inferring answers from it. The starting point of the PPAML program is that we want to develop machine-learning applications by combining probabilistic models and inference techniques. On one hand, a probabilistic model is a mathematical description of the world that expresses what we are interested in and what we are uncertain about. On the other hand, given a probabilistic model and observed data, inference techniques are ways to compute answers such as predictions and decisions. The perennial problem with probabilistic machine learning is that both probabilistic models and inference techniques are too hard to build and reuse.

The goal of the PPAML program is to use probabilistic programming to make it easier to apply machine learning and to come up with new applications [2]. As DARPA stated, achieving this goal requires not only making modeling languages more expressive and inference solvers more efficient, but also designing usable tools and infrastructure so that the expressivity and efficiency work with each other rather than against each other. Thus, central to the PPAML program is this question: how can we make probabilistic models and inference techniques easier to build and reuse as *separate* software artifacts?

Given that probabilistic models are hard to build and reuse, one response is to develop a universal and general-purpose model that can be used most of the time—the one model to end them all (such as CrossCat).

But another popular response is to make models easier to compose, out of building blocks that comprise a modeling language. That is the baseline approach taken in probabilistic programming, and by now many building blocks for modeling are well established (such as monadic bind, scoring

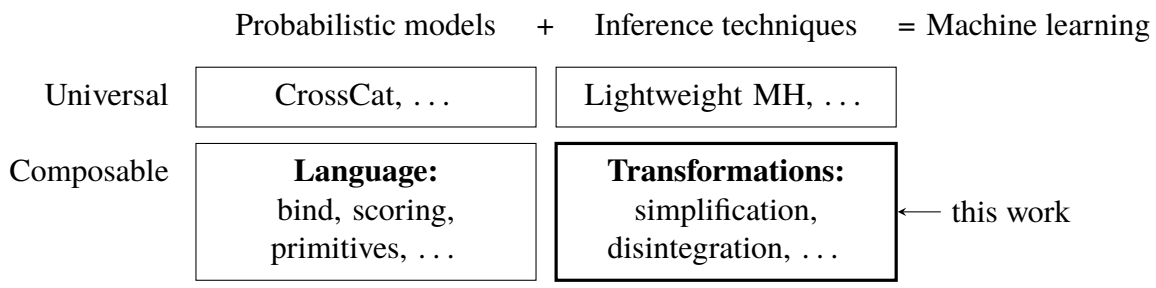



**Figure 1. Two approaches to modeling and to inference**

functions, and primitive distributions). These building blocks can be used not only to write probabilistic programs manually, but also to generate them automatically.

As for inference techniques, again they are hard to build and reuse, but here the baseline response from probabilistic programming is to build a universal and general-purpose inference algorithm that can be used most of the time—the one inference algorithm to end them all (such as single-site MH sampling over execution traces).

### 2.2 Our Work

Instead of developing a single inference algorithm, our work makes it possible to compose inference techniques out of building blocks. Because this goal is new, there is no established approach, but it is an important research program because whatever inference building blocks we come up with can be used not only by humans to express the inference algorithm they want, but also by machines to populate a search space, so we contribute to the goal of universal and general-purpose inference after all.

In the short term, our technology uniquely enables human experts to express a family of similar inference algorithms and apply them to a family of similar models. We can mix and match without reimplementing anything. Just to take one example, we can switch between Naive Bayes and Latent Dirichlet Allocation (LDA) models for text classification, and decide to apply collapsed Gibbs sampling or MH sampling with different proposal distributions, without redoing any math or rewriting any code. And although our main goal is composable reuse, our performance is also good because we can use specialized inference techniques.

Our effort in the PPAML program, leading to this new technology and its dissemination, is documented in the rest of this section.

**Composable inference** We discovered expressing each inference technique as a step in a pipeline of composable program transformations [3]. To make this discovery, we

- worked out detailed steps for several examples by hand based on challenge problems;
- created a combinator library for sequential and parallel MCMC samplers [4].

Along the way, we also

- developed a probabilistic interpreter that performs MH inference incrementally [5];
- created an extensible visualization system that allows debugging and profiling arbitrary samplers graphically.

**Disintegration transformation** We discovered specifying disintegration by a denotational equation and deriving its implementation as a program transformation [6]. To make this discovery, we

- tried expressing conditioning as sequential input from a database of random choices;
- generalized both density calculation and conditioning to disintegration;
- combined exact and approximate density calculators [7].

Extending this discovery, we also

- derived an experimental disintegrator that allows a variety of base measures;
- developed an experimental disintegrator that takes advantage of computer algebra.

**Simplification transformation** We discovered simplifying probabilistic programs by using computer algebra judiciously to simplify the *patently linear expressions* they denote [8]. To make this discovery, we

- tried simplifying density expressions by piling on rewrites, before settling on simplifying patently linear expressions instead;
- tried reasoning about random variables' domains using logical assumptions, before settling on also using regular chains.

Extending this discovery, we also rudimentarily

- interfaced simplification with Anglican;
- implemented simplification using the free library SymPy for computer algebra instead of the proprietary system Maple.

**Language design** We implemented our language and type system as a *deep embedding* [9] and designed a practitioner-friendly syntax for it. Before settling on this design, we tried implementing our language and type system as a *shallow embedding* and a *finally-tagless embedding* instead.

We provided our program transformations as function-like constructs (that is, macros) in the syntax of our language. Before settling on this design, we tried providing program transformations as command-line tools instead.

**Big data** To handle arbitrarily-large data, we added support for *arrays* to

- our language and type system,
- the expectation transformation,
- the disintegration transformation [10],
- the simplification transformation, and a trio of code-generation backends (through Haskell and C and Low Level Virtual Machine (LLVM)) [11].

We also invented a new *histogram* optimization, which improves the asymptotic time complexity of loops that arise from simplifying mixture models.

Before settling on handling arbitrarily-large data using flat arrays represented using element indices, we tried handling arbitrarily-large data using recursion and formulated a corresponding fixpoint conjecture.

**Challenge problems** We developed our team challenge problem, document classification, to emphasize modularity among models as well as inference procedures. We used our evolving system to solve this and several other challenge problems.

**Dissemination** Besides publishing the papers cited above, we also gave invited talks at

- Mathematical Foundations of Programming Semantics (MFPS) 2014 and 2016;
- Quantitative Aspects of Programming Languages and Systems (QAPL) 2016.

We also organized workshops on probabilistic programming semantics (PPS) colocated with the Symposium on Principles of Programming Languages (POPL) 2016 and 2017.

## 3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

The key enabling technology we created is a unifying language of distributions and an arsenal of automatic transformations that take programs in this language as input as well as produce them as output. These transformations constitute a calculator, except instead of calculating on numbers, they calculate on distributions. Moreover, by performing *exact* computation even on continuous distributions, they can transform models into *approximate* inference algorithms. This distribution calculator allows inference algorithms to be not only described succinctly but also executed efficiently, so it brings together people like statisticians and linguists to share their work as executable documentation, compose it as reusable modules, and collaborate with each other at a distance.

In order to invent composable inference building blocks, we examined how human practitioners explain inference techniques to each other. These explanations are typically found in Section 3 of machine learning papers—because Section 1 is the introduction and Section 2 is the model—as well as in textbooks and tutorials. It turns out that what we found can be summarized at a high level by carefully interpreting each operation in Bayes's rule:

$$\Pr(A|B) = \frac{\Pr(B|A) \times \Pr(A)}{\Pr(B)} \tag{1}$$

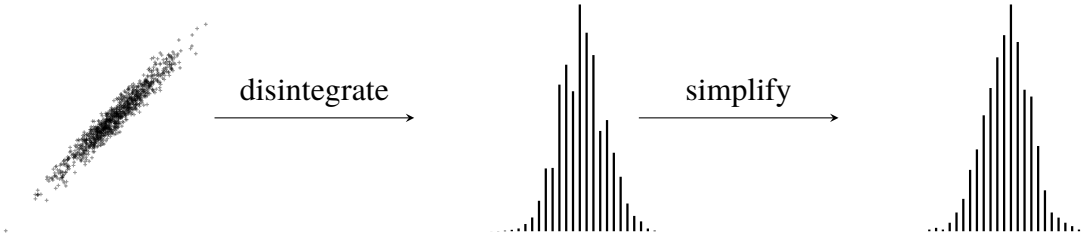
In principle, Bayes's rule tells us how to turn our prior belief about the world $\Pr(A)$, which is uninformed by observation, into our posterior belief about the world $\Pr(A|B)$, which is informed given observed data. This formula looks like it is just multiplying and dividing some numbers, but actually we are operating on distributions rather than numbers:

- What looks like multiplication $\times$ on numbers is actually *monadic bind*, an operation on distributions.
- What looks like division $\div$ on numbers is actually *disintegration*, another operation on distributions.
- What looks like equality $=$ on numbers is actually an operation that often has to turn one representation of a distribution to another (from a density to a sampler, say) using calculus.

We have automated these operations by building on programming-language and computer-algebra research. In the remainder of this section, we first illustrate these operations using a small example, then describe how they enable composable inference in more realistic applications.

### 3.1 A Small Example of Exact Inference

The following small example demonstrates the typical *composition* of inference transformations that turns a model in Hakaru into an exact solution.




(a) A scatter plot of 1000 samples from the prior (2)

(b) A histogram of 1000 weighted samples from the posterior (3)

(c) A histogram of 1000 samples from the simplified posterior (4)

**Figure 2. Operational interpretations of the programs in Section 3.1**

We begin with a prior distribution:

$$\begin{array}{l}\text{def } \mathit{prior}:\\ \quad x \leftsquigarrow \text{normal}(0, \sigma_1)\\ \quad \varepsilon \leftsquigarrow \text{normal}(0, \sigma_2)\\ \quad \text{return } (x,\ x + \varepsilon)\end{array} \tag{2}$$

One way to understand this program is *operationally*, as a procedure for generating samples randomly: on each run, the program draws two numbers $x$ and $x + \varepsilon$ independently from Gaussian distributions determined by the constants $\sigma_1$ and $\sigma_2$, then returns a pair of numbers $(x,\ x + \varepsilon)$. The result is depicted in Figure 2a. Another way to understand the same program is *denotationally*, as a two-dimensional Gaussian distribution: the first dimension $x$ represents a latent quantity we want to infer, and the second dimension $x + \varepsilon$ represents a correlated quantity we observe. Thus the monadic bind construct, notated by $\leftsquigarrow$, enjoys both an operational and a denotational interpretation. In both interpretations, we can regard $\varepsilon$ as noise that obscures $x$ from measurement.

We turn this joint distribution into a conditional distribution by applying the disintegration transformation [6]. The result is a function from the observed quantity to a measure over the latent quantity:

$$\begin{array}{l}\text{def } \mathit{posterior}(y):\\ \quad x \leftsquigarrow \text{normal}(0, \sigma_1)\\ \quad \text{factor } \dfrac{\exp\left(-(y - x)^2/2\sigma_2^2\right)}{\sqrt{2\pi}\sigma_2}\\ \quad \text{return } x\end{array} \tag{3}$$

The "factor" keyword above is common among probabilistic languages. Operationally, it equips the current sample with an importance weight or likelihood score. Any statistical analysis of the samples, such as the histogram in Figure 2b, must take these weights into account in order to be meaningful. Denotationally, it scales the measure by a factor or multiplies it by a density. Both operationally and denotationally, this program expresses our updated belief about the latent quantity $x$ given an observed value $y$ of $x + \epsilon$.

We make this conditional distribution more efficient and more perspicuous by applying the simplification transformation [8]. The resulting function looks different but denotes the same thing:

$$\begin{array}{l}\text{def } \mathit{posterior}(y)\text{:}\\ \quad \text{factor } \cdots\\ \quad \text{normal}\left(\dfrac{\sigma_1^2 y}{\sigma_1^2+\sigma_2^2}, \dfrac{\sigma_1\sigma_2}{\sqrt{\sigma_1^2+\sigma_2^2}}\right)\end{array} \tag{4}$$

Given an observed value $y$, this program produces all samples with the same importance weight, rather than a random mix of important and unimportant samples. Hence, any statistical analysis of the samples, such as the smoother histogram in Figure 2c, would be less subject to the vagaries of random variation than before simplification. But in this example, to understand the posterior distribution, we do not need any samples from it, because it turns out to be exactly proportional to a new Gaussian distribution whose parameters are solved in closed form at the bottom of (4) by the simplification transformation. Those parameters, such as the new mean $\sigma_1^2 y/(\sigma_1^2+\sigma_2^2)$, can be either read off by syntactic inspection or extracted as moments by the expectation transformation [7].

### 3.2 Larger Models and Approximate Inference

Having demonstrated the most important Hakaru transformations performing exact inference on a small model, we describe how they also enable approximate inference on larger models.

The small model in Section 3.1 generalizes from one-dimensional Bayesian linear regression with one data point to multi-dimensional Bayesian linear regression with many data points. Even if the number of dimensions and the number of data points are unknown, because the Hakaru language and its transformations support symbolic arrays and loops [10], the same workflow delivers the exact posterior distribution in closed form. In general, the composition of disintegration followed by simplification produces efficient solvers from those generative models that a human practitioner can solve exactly by hand using conjugacy relationships [9].

The vast majority of models do not admit exact solutions in closed form. In those cases, disintegration followed by simplification produces a representation of the posterior distribution that invokes a scoring function. Without Hakaru, the typical practitioner would proceed to derive and implement an approximate inference algorithm using techniques such as Markov chain Monte Carlo (MCMC) or variational inference. The goal of our research is to automate the mathematics and programming required to turn a posterior distribution into an approximate inference algorithm.

It is well known that computer mathematics and program transformations can help automate inference techniques. For example, automatic differentiation can help automate maximum likelihood and variational inference. Our research extends this knowledge to approximate inference techniques whose automation requires calculating with distributions. It turns out that many MCMC techniques are defined in the literature in terms of the very operations performed by our transformations—not applied to distributions that represent our belief about the world, but to distributions that represent the approximation process and ensure its correctness mathematically.

- For example, in *MH sampling* [12, 13], the acceptance ratio is computed by applying disintegration followed by expectation to the target and proposal distributions [14].

- And in *Gibbs sampling* [15, 16], the transition kernel is computed by applying disintegration followed by simplification to the target distribution.

Accordingly, we have used our program transformations to mechanize these definitions and produce MH and Gibbs samplers automatically [9, 3]. These models are described in Chapter 4 below.

In summary, whether an exact solution is available in closed form, our program transformations let us succinctly describe and execute the entire pipeline from the probabilistic model to the inference algorithm. But it is when we produce an efficient approximation algorithm by applying disintegration and simplification multiple times that our composable building blocks really shine in their reusability.

## 4.0 RESULTS AND DISCUSSION

We describe how 6 probabilistic models are turned automatically into inference algorithms by Hakaru transformations, advancing the state of the art in terms of modularity and performance. These benchmarks and results are summarized in broad strokes in Table 1.

**Table 1. Benchmarks and results summary**

| **Model** | **Data** | **Inference** | **Rough comparison** | | | |
|---|---|---|---|---|---|---|
| | | | Baseline | Modularity | Speed | Accuracy |
| Linear regression | Synthetic | Exact | Handwritten | more | same | same |
| Clinical trial | Synthetic | Exact | Handwritten | more | same | same |
| Linear dynamics | Synthetic | MH | WebPPL | same | $4\times$ | $10\times$ |
| Gaussian mixture | Synthetic | Gibbs | JAGS | same | $1/10\times$ | $3\times$ |
| Naive Bayes | 20 Newsgroups | Gibbs | JAGS | same | same | $10\times$ |
| LDA | 20 Newsgroups | Gibbs | MALLET | more | $1/2\times$ | same |

The first 2 models are amenable to exact inference. Like in Section 3.1, our disintegration and simplification transformations turn the models into exact posterior distributions, in the same closed form that a human practitioner would derive and implement by hand. The code we generate is as fast and as accurate as handwritten, but more modular in the sense that the same transformations automatically handle different models.

The 4 remaining models [3] call for approximate inference, typically MCMC. But as human practitioners know, approximations like MCMC achieve higher accuracy in fewer iterations—and become less subject to the vagaries of random variation—when as many latent random variables as possible are first *collapsed* (or *eliminated*, or *integrated out*, or *Rao-Blackwellized*) by exact inference. Hakaru transformations constitute the first probabilistic programming system that automates this exact inference. Moreover, we can compose exact and approximate inference techniques for different models. So compared to other probabilistic programming systems such as WebPPL and JAGS, the samplers we generate are more accurate. And compared to specialized tools such as the text-processing toolkit MALLET, our sampler is more modular.

## 5.0 CONCLUSIONS

We have identified a compact suite of operations on probability distributions that people use to explain inference techniques to each other. We have automated these operations as program transformations, so that people can not only compose them to give succinct explanations but also reuse them to perform efficient inference.

Our code implementing these transformations is freely available (`https://github.com/hakaru-dev/hakaru`). However, this code is not the only way we've disseminated our work, and neither should it be, because our approach is not tied to a particular programming language. Many other probabilistic-programming developers want to re-implement our definitions in their systems rather than invoke our code. So it is important that we have also published papers [6, 8, 7, 10, 9, 3], given talks, and worked across institutions to help people replicate our building blocks.

## 6.0 RECOMMENDATIONS

The initial success of our research program, both in reducing human effort and in improving solver speed and accuracy, suggests that it would be profitable to generalize our language and transformations to automate more workflows that turn probabilistic models into inference algorithms. The most promising generalizations proceed along three dimensions.

First, the transformations should handle more distributions:

- The Hakaru language can express arrays and the disintegration transformation can observe them, but the current disintegration transformation returns no result if asked to observe part of an array. Such observations are useful for generating Gibbs samplers. Disintegration should also handle arrays whose elements are generated by a mixture of control paths.

- The Hakaru language can express mixtures of discrete and continuous distributions, but the current disintegration transformation returns no result if asked to observe them. Such observations are useful for handling censored measurements (such as an underexposed or overexposed photograph) and for generating *single-site* MH samplers [17].

- The Hakaru language can express Markov chains, but when the length of the chain is large or unknown, the current simplification transformation does not use the forward-backward algorithm to collapse the hidden states of a hidden Markov model efficiently [18].

- The simplification transformation recognizes a primitive distribution by converting a density function to its *holonomic representation* [8], but the Hakaru language only expresses primitive distributions in a finite set of families such as Gaussian. Extending Hakaru to express all holonomic densities would enable efficient execution of more simplification results.

Second, the language should support more data types:

- *Infinite arrays* would be useful for expressing nonparametric models. For example, random infinite arrays generated by independent choices can express Dirichlet processes.

- *Trees* would be useful for expressing syntactic structures, such as parses generated by a probabilistic context-free grammar.

- *Functions* in a restricted sense would be useful for expressing continuous models of time and space, such as Gaussian processes.

Finally, there are reasons to expand our modest repertoire of transformations operating on Hakaru programs:

- *Optimizing* an objective function, with respect to the parameters of a distribution, is a useful building block even though it is not exactly Bayesian. For example, maximum-likelihood estimation is optimization. Optimization can be performed by exact computation, (stochastic) gradient descent, expectation maximization, or simulated annealing.

- *Reparameterizing* a distribution, so that it is expressed as an invertible transformation of another distribution, can ease understanding as well as inference. In particular, reparameterization often enables *variational inference*, a form of optimization.

- As more transformations become available and applicable to larger programs and their parts, it becomes a pressing question how to specify robustly when and where to apply transformations. One potential answer is an interactive term-rewriting assistant.

## REFERENCES


[1] George Casella and Christian P. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

[2] Probabilistic programming for advancing machine learning. Broad Agency Announcement DARPA-BAA-13-31, DARPA, 2013.

[3] Robert Zinkov and Chung-chieh Shan. Composing inference algorithms as program transformations. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, Corvallis, Oregon, 2017.

[4] Praveen Narayanan and Chung-chieh Shan. A combinator library for MCMC sampling. Poster at the NIPS workshop on probabilistic programming, 2014.

[5] Oleg Kiselyov. Probabilistic programming language and its incremental evaluation. In Atsushi Igarashi, editor, *Proceedings of APLAS 2016: 14th Asian Symposium on Programming Languages and Systems*, number 10017 in Lecture Notes in Computer Science, pages 357–376, Berlin, 21–23 November 2016. Springer.

[6] Chung-chieh Shan and Norman Ramsey. Exact Bayesian inference by symbolic disintegration. In *POPL '17: Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pages 130–144, New York, January 2017. ACM Press.

[7] Wazim Mohammed Ismail and Chung-chieh Shan. Deriving a probability density calculator (functional pearl). In Jacques Garrigue, Gabriele Keller, and Eijiro Sumii, editors, *ICFP '16: Proceedings of the ACM International Conference on Functional Programming*, pages 47–59, New York, 2016. ACM Press.

[8] Jacques Carette and Chung-chieh Shan. Simplifying probabilistic programs using computer algebra. In Marco Gavanelli and John H. Reppy, editors, *Practical Aspects of Declarative Languages: 18th International Symposium, PADL 2016*, number 9585 in Lecture Notes in Computer Science, pages 135–152, Berlin, 2016. Springer.

[9] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in Hakaru (system description). In Oleg Kiselyov and Andy King, editors, *Proceedings of FLOPS 2016: 13th International Symposium on Functional and Logic Programming*, number 9613 in Lecture Notes in Computer Science, pages 62–79, Berlin, 4–6 March 2016. Springer.

[10] Praveen Narayanan and Chung-chieh Shan. Symbolic conditioning of arrays in probabilistic programs. *Proceedings of the ACM on Programming Languages*, 1(ICFP):11:1–11:25, 2017.

[11] Rajan Walia, Jacques Carette, Praveen Narayanan, Chung-chieh Shan, and Sam Tobin-Hochstadt. Efficient compilation of array probabilistic programs. Submitted, 2017.

[12] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[13] W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[14] Luke Tierney. A note on Metropolis-Hastings kernels for general state spaces. *The Annals of Applied Probability*, 8(1):1–9, February 1998.

[15] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[16] Alan E. Gelfand, Adrian F. M. Smith, and Tai-Ming Lee. Bayesian analysis of constrained parameter and truncated data problems using Gibbs sampling. *Journal of the American Statistical Association*, 87(418):523–532, 1992.

[17] David Wingate, Andreas Stuhlmüller, and Noah D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of AISTATS 2011: 14th International Conference on Artificial Intelligence and Statistics*, number 15 in JMLR Workshop and Conference Proceedings, pages 770–778, Cambridge, 11–13 April 2011. MIT Press.

[18] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

## LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

| | |
|---|---|
| JAGS | Just Another Gibbs Sampler |
| LDA | Latent Dirichlet Allocation |
| LLVM | Low Level Virtual Machine |
| MALLET | MAchine Learning for LanguagE Toolkit (including document-classification tools) |
| MCMC | Markov chain Monte Carlo |
| MFPS | Mathematical Foundations of Programming Semantics |
| MH | Metropolis-Hastings |
| POPL | Symposium on Principles of Programming Languages |
| PPAML | Probabilistic Programming for Advancing Machine Learning |
| PPS | Workshop on Probabilistic Programming Semantics |
| QAPL | Quantitative Aspects of Programming Languages and Systems |
| WebPPL | Web Probabilistic Programming Language (embedded in JavaScript) |